Problem Set 2 Knitted

Jiaxuan Liu

2024-10-18

Problem Set 2 Report

Introduction

1. "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **JL**
2. "I have uploaded the names of anyone I worked with on the problem set here" asdgla ;sdgnawe;oe
3. Late coins used this pset: ***1***

Data cleaning continued

Q1: Counting NA values by defining function

For each column, how many rows are NA? Write a function that returns a two-column DataFrame where each row is a variable, the first column of the DataFrame is the name of each variable, and the second column of the DataFrame is the number of times that the column is NA.

Importing the dataset

```
import matplotlib.pyplot as plt
import jupyter
import pandas as pd
```

```
# File path for the parking tickets dataset
file_path = '/Users/doudouliu/Desktop/python 2/student30538/problem_sets/ps2/data/parking_ti

# Reading the CSV file
df = pd.read_csv(file_path)

# Display first few rows to ensure the dataset is loaded correctly
print(df.head())
```

```
   Unnamed: 0  ticket_number           issue_date violation_location  \
0           1     51482901.0  2007-01-01 01:25:00    5762 N AVONDALE
1           2     50681501.0  2007-01-01 01:51:00    2724 W FARRAGUT
2           3     51579701.0  2007-01-01 02:22:00       1748 W ESTES
3           4     51262201.0  2007-01-01 02:35:00    4756 N SHERIDAN
4           5     51898001.0  2007-01-01 03:50:00    7134 S CAMPBELL

                             license_plate_number license_plate_state  \
0  d41ee9a4cb0676e641399ad14aaa20d06f2c6896de6366...                  IL
1  3395fd3f71f18f9ea4f0a8e1f13bf0aa15052fc8e5605a...                  IL
2  302cb9c55f63ff828d7315c5589d97f1f8144904d66eb3...                  IL
3  94d018f52c7990cea326d1810a3278e2c6b1e8b44f3c52...                  IL
4  876dd3a95179f4f1d720613f6e32a5a7b86b0e6f988bf4...                  IL

  license_plate_type      zipcode violation_code  \
0                PAS  606184118.0       0964090E
1                PAS  606454911.0       0964090E
2                PAS  604116803.0       0964150B
3                PAS  606601345.0       0976160F
4                PAS  606291432.0       0964100A

                    violation_description  ...  fine_level2_amount  \
0                RESIDENTIAL PERMIT PARKING  ...                 100
1                RESIDENTIAL PERMIT PARKING  ...                 100
2       PARKING/STANDING PROHIBITED ANYTIME  ...                 100
3  EXPIRED PLATES OR TEMPORARY REGISTRATION  ...                 100
4                WITHIN 15' OF FIRE HYDRANT  ...                 200

  current_amount_due total_payments  ticket_queue  ticket_queue_date  \
0                0.0           50.0          Paid         2007-03-20
1                0.0           50.0          Paid         2007-01-31
2              122.0            0.0        Notice         2007-02-28
3                0.0           50.0          Paid         2007-01-11
```

```
4                0.0          100.0          Paid         2007-04-25

   notice_level  hearing_disposition notice_number officer  \
0          DETR               Liable  5.080059e+09   17266
1          VIOL                  NaN  5.079876e+09   10799
2          SEIZ                  NaN  5.037862e+09   17253
3           NaN                  NaN  5.075310e+09    3307
4          DETR                  NaN  5.073568e+09   16820

                        address
0  5700 n avondale, chicago, il
1  2700 w farragut, chicago, il
2     1700 w estes, chicago, il
3  4700 n sheridan, chicago, il
4  7100 s campbell, chicago, il

[5 rows x 24 columns]


/var/folders/b0/6kybjmv57sv8drq7v4kr7w240000gn/T/ipykernel_2628/3451754831.py:9: DtypeWarning
  df = pd.read_csv(file_path)
```

How many NAs

```
def Nna(df):
  na_N = df.isna().sum().reset_index()  # convert to DataFrame format with NA counts
  na_N.columns = ['variable', 'na_count']  # rename columns
  return na_N

print (Nna(df))
```

```
                variable  na_count
0             Unnamed: 0         0
1          ticket_number         0
2             issue_date         0
3     violation_location         0
4   license_plate_number         0
5    license_plate_state        97
6     license_plate_type      2054
7                zipcode     54115
```

8	violation_code	0
9	violation_description	0
10	unit	29
11	unit_description	0
12	vehicle_make	0
13	fine_level1_amount	0
14	fine_level2_amount	0
15	current_amount_due	0
16	total_payments	0
17	ticket_queue	0
18	ticket_queue_date	0
19	notice_level	84068
20	hearing_disposition	259899
21	notice_number	0
22	officer	0
23	address	0

Q2 Three variables are missing much more frequently than the others. Why? (Hint: look at some rows and read the data dictionary written by ProPublica)

Hearing_disposition has a missing vlaue of 259899, given that the contested tickets that got taken to the court and make it further to the hearing stage is rare, most tickets are either not contested or got resolved earlier, the large number of missing values make sense.

Notice_level has a missing value of 84068, given that this column measures the tickets number that reaches the final stage (i.e. breaching, violating), the tickets that are paid off or dismissed takes up the majority of all tickets, which justify the large number of na.

zipcode takes up 54115 missing values, this is due to the data collection process, given the limitation of technology to track to location of entries.

Q3

```
import bs4
import requests
from bs4 import BeautifulSoup
import re

url = 'https://www.chicago.gov/city/en/depts/fin/provdrs/parking_and_redlightcitationadminist

seed = requests.get(url)
soup = BeautifulSoup(seed.text, 'html.parser')
table = soup.find('table')
rows = table.find_all('tr')


for row in rows:
```

```
    # Extract columns for each row
    cols = row.find_all('td')
    cols_text = [col.text.strip() for col in cols]

    # Check if any column contains the word 'sticker'
    if any('sticker' in col.lower() for col in cols_text):
        print(cols_text)
```

```
['9-64-125(b)', 'No City Sticker Vehicle Under/Equal to 16,000 LBS', '$200.00', '$50.00']
['9-64-125(c)', 'No City Sticker Vehicle Over 16,000 LBS', '$250.00', '$0.00']
['9-64-125(d)', 'Improper Display of City Sticker', '$30.00', '$30.00']
```

By scraping the table, we have Code 9-64-125 for old sticker misplacement or missing as the violation code, the new violation code is Code 9-64-125(b), which differentiate the penalities by the weight of vehicle

Q4

```
url = 'https://www.propublica.org/article/chicago-vehicle-sticker-law-ticket-price-hike-blac

seed = requests.get(url)
soup = BeautifulSoup(seed.text, 'html.parser')

paragraphs = soup.find_all('p')

text = soup.get_text()
numbers = re.findall(r'\b\d+\b', text)

# locate all the numbers in the same line with buzz words like "violation", "fine"
for p in paragraphs:
    text = p.get_text()
    if "violation" in text.lower() or "code" in text.lower() or "sticker" in text.lower() an
        print(text)
        # Find and print numbers in the paragraph
        paragraph_numbers = re.findall(r'\b\d+\b', text)
        if paragraph_numbers:
            print(f"Numbers found: {paragraph_numbers}")
```

```
During negotiations for Chicago's 2012 budget, newly elected Mayor Rahm Emanuel and then-City
Numbers found: ['2012', '120', '200']
She suggested instead that the city raise penalties for sticker "scofflaws." Aldermen applauc
In an interview, Mendoza said the final sticker ticket price "was based on the fact that the
```

After anchoring for key words like "violation" and "sticker", we can locate the paragraph which indicates that the old sticker price and the new are 120 and 200 (or 250, depending on the weight, but we don't have to worry about that) respectively.

Revenue increase from "missing city sticker" tickets

Q1

```
#check the format
df['violation_code'].unique()
df.head()

# the codes are stored as numeric value instead of string, so no "-" or "()" are shown as der
# 0964125', '0964125B are not combined in one value "0964125_combined"
def combine_violation_codes(code):
    if code in ['0964125', '0964125B']:
        return '0964125_combined'
    return code

df['violation_code_combined'] = df['violation_code'].apply(combine_violation_codes)

df.head()

#Now collapse the new created value, first we need to convert datatime format
df['issue_date'] = pd.to_datetime(df['issue_date'])
df_sticker = df[df['violation_code_combined'] == '0964125_combined']

monthly_ticket = df_sticker.resample('M', on='issue_date').size().reset_index(name='count_of_
print (monthly_ticket)

#ALtiar plotting
import altair as alt
```

```
chart = alt.Chart(monthly_ticket).mark_line().encode(
    x='issue_date:T',  # Time on the x-axis from 'monthly_ticket'
    y='count_of_combined_violations:Q',  # Quantitative count on the y-axis from 'monthly_ti
    tooltip=['issue_date:T', 'count_of_combined_violations:Q']  # Tooltip for more details o
).properties(
    title='Number of Combined Violation Tickets Over Time'
)

chart.show()
```

```
    issue_date  count_of_combined_violations
0   2007-01-31                           160
1   2007-02-28                           104
2   2007-03-31                           161
3   2007-04-30                           158
4   2007-05-31                           126
..         ...                           ...
132 2018-01-31                           168
133 2018-02-28                           119
134 2018-03-31                           171
135 2018-04-30                           153
136 2018-05-31                            79

[137 rows x 2 columns]
```

```
/var/folders/b0/6kybjmv57sv8drq7v4kr7w240000gn/T/ipykernel_2628/1938627337.py:20: FutureWarni
  monthly_ticket = df_sticker.resample('M', on='issue_date').size().reset_index(name='count_
```

```
alt.Chart(...)
```

Q2

after reseraching stack overflow, the relevant help page is Altair Axis Document. We are essentially creatnig another altair chart with a more precise x-axis label

```
import altair as alt

# Create an Altair line chart with customized x-axis date labels
chart = alt.Chart(monthly_ticket).mark_line().encode(
    x=alt.X('issue_date:T',
            axis=alt.Axis(
                format='%Y-%m',  #high light Year-Month
                labelAngle=-45,  # rotate labels for better fit
                tickCount='month'
            )),
    y='count_of_combined_violations:Q',
    tooltip=['issue_date:T', 'count_of_combined_violations:Q']
).properties(
    title='Number of Combined Violation Tickets Over Time with Custom Date Labels'
)

chart.show()
```

```
alt.Chart(...)
```

Q3 # Given that the increase in the penality happen in year 2012, the increase in revenue for each ticket is 80 dollars (200 - 120), and the sample is 1% of the population, we can calculate the following. The increase in revenue should be $15,360,000

```
# Given the increase in the vehicle city sticker violation fine in Chicago occurred in 2012,
year_begin = '2011-02-24'
Year_end = '2012-02-24'

df_before_after = df_sticker[(df_sticker['issue_date'] >= year_begin) & (df_sticker['issue_da

#we then count the number in the filter data
tickets_1p = df_before_after.shape[0]

# scale up by 100 to infer population tickets
tickets_100p = tickets_1p * 100

# The increase in price is 200-120 = 80
revenue_increase = tickets_100p * 80

print(f"Revenue increase from new penality is ${revenue_increase:,}")
```

```
Revenue increase from new penality is $15,360,000
```

Q4

The wording of this question is a little vague, but from what i understand, the denominator of the repayment rate is the numbeR of tickets with both some partial payments and those that are paid up. The following operation is base on this assumption.

The repayment rate drops from 49.08% to 43.37%.

```
# Filter tickets between 2012-02-25 and 2013-02-25
year_after = df_sticker[(df_sticker['issue_date'].dt.date.between(pd.to_datetime('2012-02-25

# For year_1 (2012)
paid_ticket_1 = year_after[year_after['total_payments'] > 0]  # filter tickets with payments
total_tickets_after = year_after['ticket_number'].nunique()  # total tickets issued before ir
total_paid_tickets_after = paid_ticket_1['ticket_number'].nunique()  # total paid tickets
repayment_rate_after = total_paid_tickets_after / total_tickets_after  # repayment rate

# Print repayment rates
print(f"Repayment Rate for 2012-13 (After Price Increase): {repayment_rate_after * 100:.2f}%

# For year_0 (2011)
year_before = df_sticker[(df_sticker['issue_date'].dt.date.between(pd.to_datetime('2011-02-25

paid_ticket_0 = year_before[year_before['total_payments'] > 0]

total_tickets_before = year_before['ticket_number'].nunique()

total_paid_tickets_before = paid_ticket_0['ticket_number'].nunique()

repayment_rate_before = total_paid_tickets_before / total_tickets_before

print(f"Repayment Rate for 2011-12 (After Price Increase): {repayment_rate_before * 100:.2f}%
```

```
Repayment Rate for 2012-13 (After Price Increase): 43.74%
Repayment Rate for 2011-12 (After Price Increase): 49.08%
```

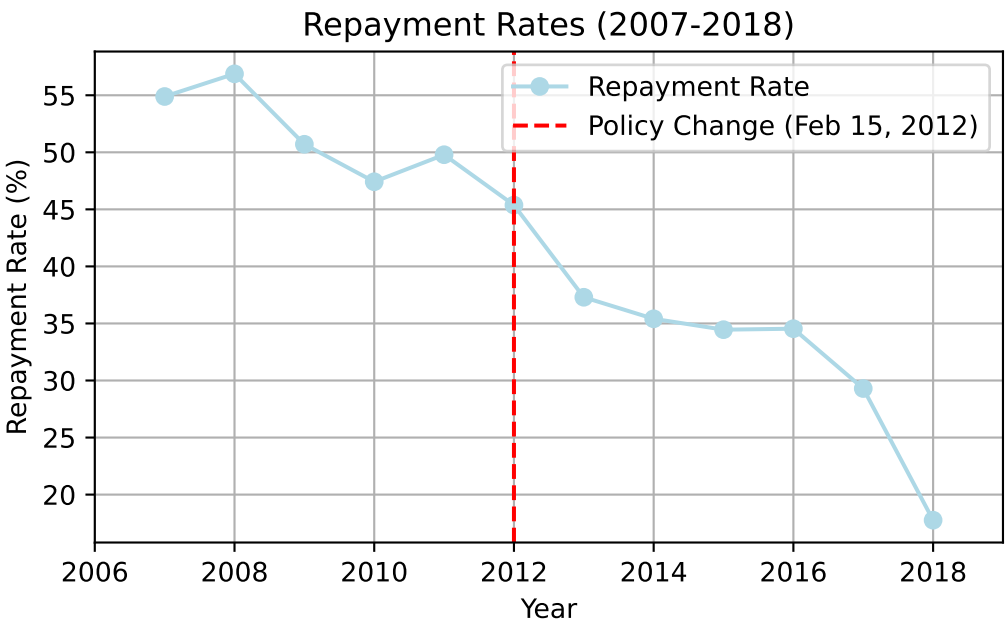
Q5

As shown in the graph, there is a clear decline following the policy change marked by the red vertical line on February 15, 2012. Before the policy, repayment rates hovered around 55%, but after 2012, there was a significant drop, with rates steadily decreasing to below 20% by 2018. This sharp decline suggests that the policy change, likely involving a price increase, negatively impacted people's ability or willingness to pay, resulting in much lower repayment rates over time.

The plot demonstrates the imporant concept in class, how the scale of one graph can be changed to give different impressions to the reader. In Q4, when I first saw the numeric change in the repayment rate, it does not appear striking; however after adjusting for the scale of Y-axis. We get this utterly steep curve.

```
# Filter data and calculate repayment rates for every year
df_filtered = df_sticker[(df_sticker['issue_date'].dt.year >= 2007) & (df_sticker['issue_date
df_filtered['year'] = df_filtered['issue_date'].dt.year
repayment_rates = df_filtered.groupby('year').apply(lambda x: (x[x['total_payments'] > 0]['t

# Plot with connected dots
plt.grid(True)
plt.plot(repayment_rates.index, repayment_rates.values, color='lightblue', marker='o', label=
plt.axvline(x=2012, color='red', linestyle='--', label='Policy Change (Feb 15, 2012)')
plt.xlim(2006, 2019)
plt.xlabel('Year'); plt.ylabel('Repayment Rate (%)'); plt.title('Repayment Rates (2007-2018)
plt.legend(); plt.tight_layout(); plt.show()
```

```
/var/folders/b0/6kybjmv57sv8drq7v4kr7w240000gn/T/ipykernel_2628/3746950398.py:4: DeprecationW
  repayment_rates = df_filtered.groupby('year').apply(lambda x: (x[x['total_payments'] > 0][
```



Q6

Based on the column ranking, which represents the product of repayment rate and the number of tickets—a simple measure that captures both aspects of revenue generation—the top three recommendations are Expired Plates or Temporary Registration, PARKING/STANDING PROHIBITED ANYTIME and RUSH HOUR PARKING

```
# Group by violation type, calculate the number of tickets and repayment rate for each

df_grouped = df.groupby(['violation_code', 'violation_description']).apply(
    lambda x: x['ticket_number'].nunique() * (x[x['total_payments'] > 0]['ticket_number'].nu
).reset_index(name='product')

# Sort to find the top three violation types based on potential revenue (product)
top_violations = df_grouped.sort_values(by='product', ascending=False).head(3)

top_violations[['violation_code', 'violation_description', 'product']]
```

```
/var/folders/b0/6kybjmv57sv8drq7v4kr7w240000gn/T/ipykernel_2628/145113783.py:3: DeprecationWa
  df_grouped = df.groupby(['violation_code', 'violation_description']).apply(
```

	violation_code	violation_description	product
101	0976160F	EXPIRED PLATES OR TEMPORARY REGISTRATION	12482.0
53	0964150B	PARKING/STANDING PROHIBITED ANYTIME	8689.0
21	0964080A	RUSH HOUR PARKING	6223.0

Headlines and sub-messages # Q1 # Make a dataframe for violation description, fraction of payment time, average level 1 fine # The top 5 are EXPIRED PLATES OR TEMPORARY REGISTRATION, STREET CLEANING, RESIDENTIAL PERMIT PARKING, EXP. METER NON-CENTRAL BUSINESS DISTRICT, PARKING/STANDING PROHIBITED ANYTIME

```python
df.head()
import pandas as pd

#Group by description
grouped = df.groupby('violation_description').agg(
    fraction_tickets_paid=('total_payments', lambda x: (x > 0).mean()),
    avg_fine_level1=('fine_level1_amount', 'mean'),
    total_tickets=('violation_description', 'size')
).reset_index()

# Rank from high to low
sorted_df = grouped.sort_values('total_tickets', ascending=False)

#top 5 and print
top_5_violations = sorted_df.head(5)
print(top_5_violations)
```

```
                      violation_description  fraction_tickets_paid  \
23   EXPIRED PLATES OR TEMPORARY REGISTRATION               0.608065
101                          STREET CLEANING               0.815896
90                RESIDENTIAL PERMIT PARKING               0.745978
19   EXP. METER NON-CENTRAL BUSINESS DISTRICT               0.795485
81        PARKING/STANDING PROHIBITED ANYTIME               0.710677

     avg_fine_level1  total_tickets
23         54.968869          44811
```

```
101         54.004249          28712
90          66.338302          23683
19          46.598058          20600
81          66.142864          19753
```

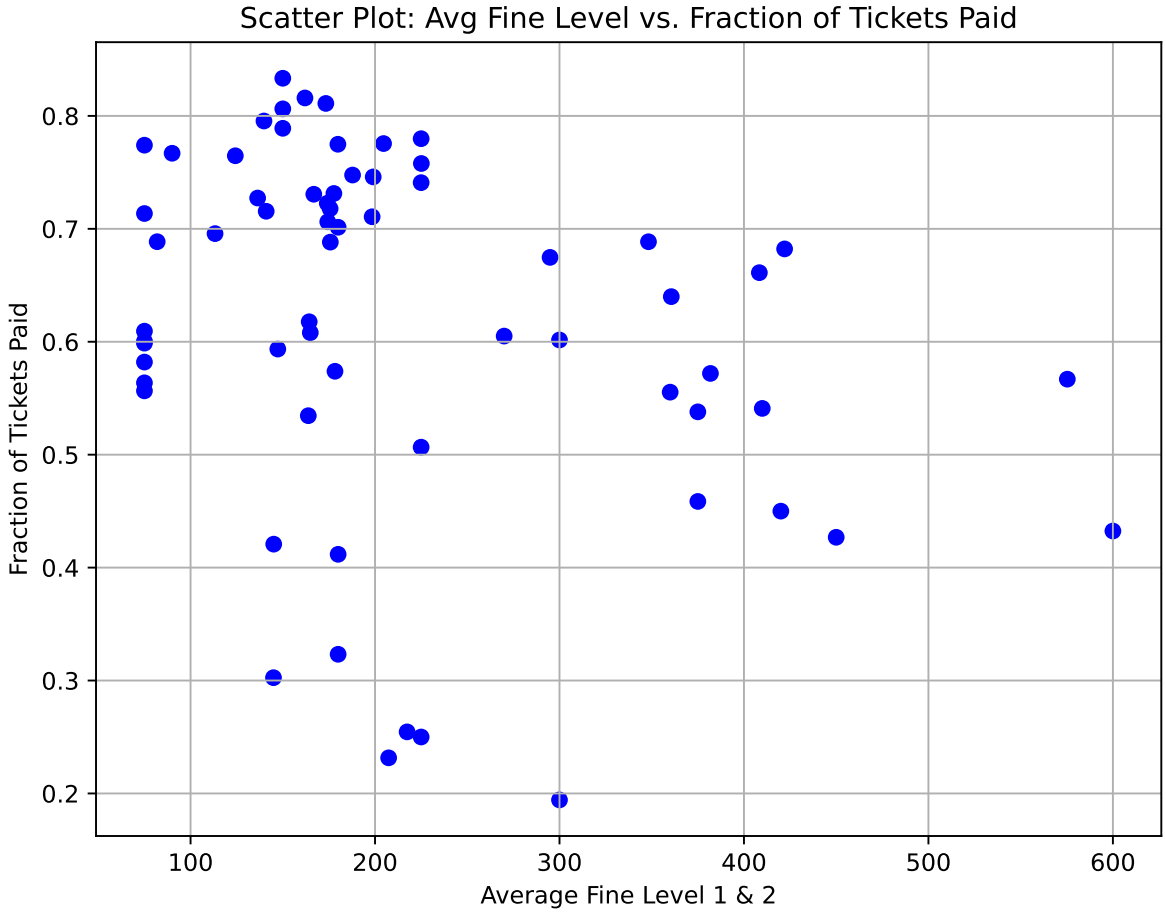
Q2

```
df_100 = df.groupby('violation_description').filter(lambda x: len(x) >= 100)

grouped2 = df_100.groupby('violation_description').agg(
    fraction_tickets_paid=('total_payments', lambda x: (x > 0).mean()),
    avg_fine_level1_2=('fine_level1_amount', lambda x: (x + df_100.loc[x.index, 'fine_level2
    total_tickets=('violation_description', 'size')
).reset_index()

print(grouped2)

#Drop the outlier
max_2 = grouped2['avg_fine_level1_2'].max()
print (max_2) #1455.34
grouped2 = grouped2[grouped2['avg_fine_level1_2'] != max_2]

#create scatter and print
plt.figure(figsize=(8, 6))
plt.scatter(grouped2['avg_fine_level1_2'], grouped2['fraction_tickets_paid'], color='blue')
plt.title('Scatter Plot: Avg Fine Level vs. Fraction of Tickets Paid')
plt.xlabel('Average Fine Level 1 & 2')
plt.ylabel('Fraction of Tickets Paid')
plt.grid(True)
plt.show()

#Alternative 1: Bar Plot
# Create bins for avg_fine_level1_2
grouped2['fine_bins'] = pd.cut(grouped2['avg_fine_level1_2'], bins=5)

# Group by bins and get the mean of the fraction of tickets paid
bin_group = grouped2.groupby('fine_bins').agg(
    avg_fraction_paid=('fraction_tickets_paid', 'mean'),
    count=('violation_description', 'size')  # Count the number of violations in each bin
).reset_index()
```

```
# Create the bar plot
plt.figure(figsize=(8, 6))
plt.bar(bin_group['fine_bins'].astype(str), bin_group['avg_fraction_paid'], color='green')
plt.title('Bar Plot: Average Fraction of Tickets Paid by Fine Bins')
plt.xlabel('Fine Level Bins')
plt.ylabel('Average Fraction of Tickets Paid')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

#Alternative 2: Box plot
# Create bins for avg_fine_level1_2 to use as categories in the box plot
grouped2['fine_bins'] = pd.cut(grouped2['avg_fine_level1_2'], bins=5)

# Create the box plot
plt.figure(figsize=(8, 6))
grouped2.boxplot(column='fraction_tickets_paid', by='fine_bins', grid=True)
plt.title('Box Plot: Distribution of Fraction of Tickets Paid by Fine Levels')
plt.suptitle('')  # Remove the automatic subtitle
plt.xlabel('Fine Level Bins')
plt.ylabel('Fraction of Tickets Paid')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

#In my opinion, the scatter plot is the most effective of the three for demonstration purpos
```

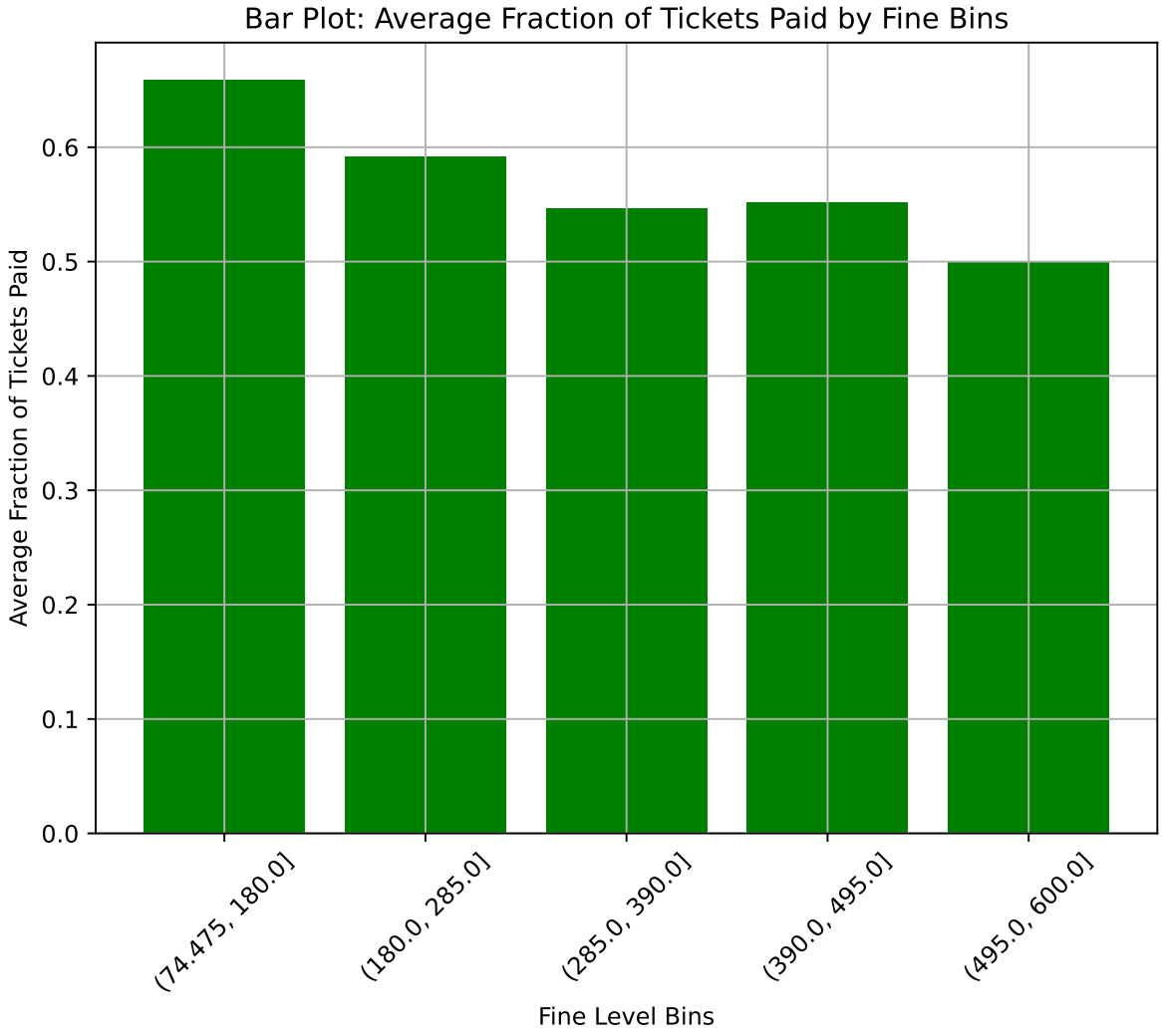
```
                     violation_description  fraction_tickets_paid  \
0                          20'OF CROSSWALK               0.722646
1                        3-7 AM SNOW ROUTE               0.730632
2    ABANDONED VEH. FOR 7 DAYS OR INOPERABLE               0.254529
3       BLOCK ACCESS/ALLEY/DRIVEWAY/FIRELANE               0.661178
4      BURGLAR ALARM SOUNDING OVER 4 MINUTES               0.764706
..                                       ...                    ...
61  TRUCK,RV,BUS, OR TAXI RESIDENTIAL STREET               0.695761
62     TWO HEAD LAMPS REQUIRED VISIBLE 1000'               0.600451
63       WINDOWS MISSING OR CRACKED BEYOND 6               0.609375
64                WITHIN 15' OF FIRE HYDRANT               0.688565
65         WRONG DIRECTION OR 12'' FROM CURB               0.774077

    avg_fine_level1_2  total_tickets
```

```
0          174.122137           393
1          166.805721           839
2          217.391304          1104
3          408.343889          1579
4          124.264706           102
..                ...           ...
61         113.275214          4789
62          75.000000           443
63          75.000000           576
64         348.238860          6104
65          75.000000          1111

[66 rows x 4 columns]
1455.3435114503816
```

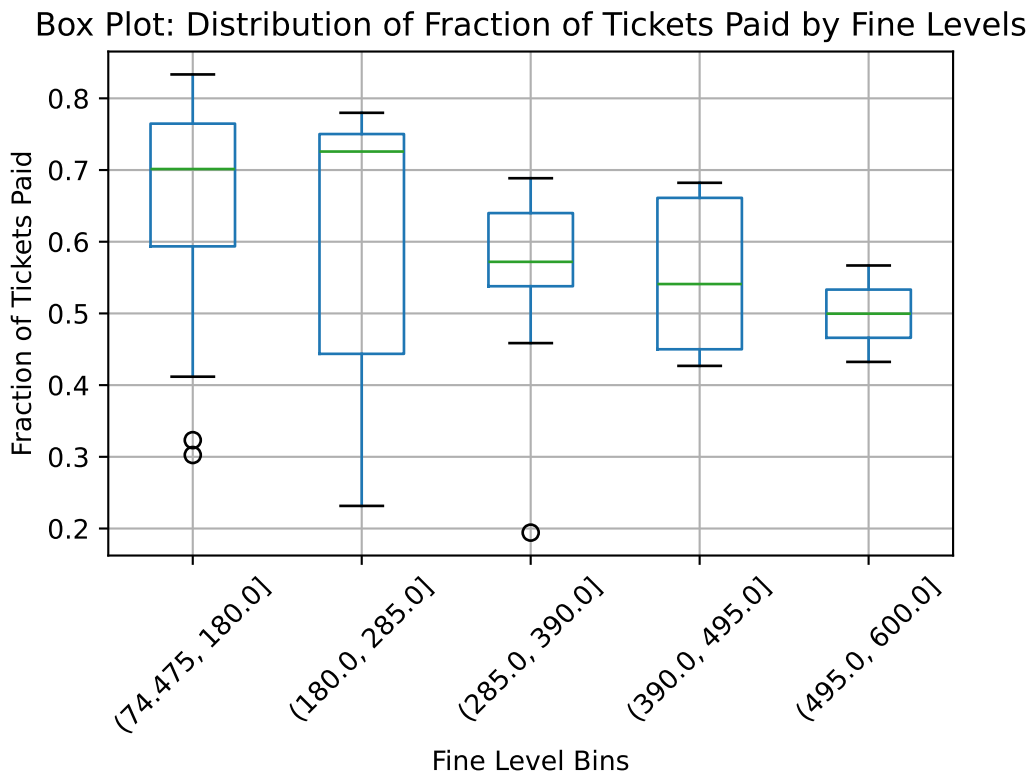



```
/var/folders/b0/6kybjmv57sv8drq7v4kr7w240000gn/T/ipykernel_2628/2759984048.py:30: FutureWarni
```

```
bin_group = grouped2.groupby('fine_bins').agg(
```




```
<Figure size 2400x1800 with 0 Axes>
```



Understanding the structure of the data and summarizing it

Q1

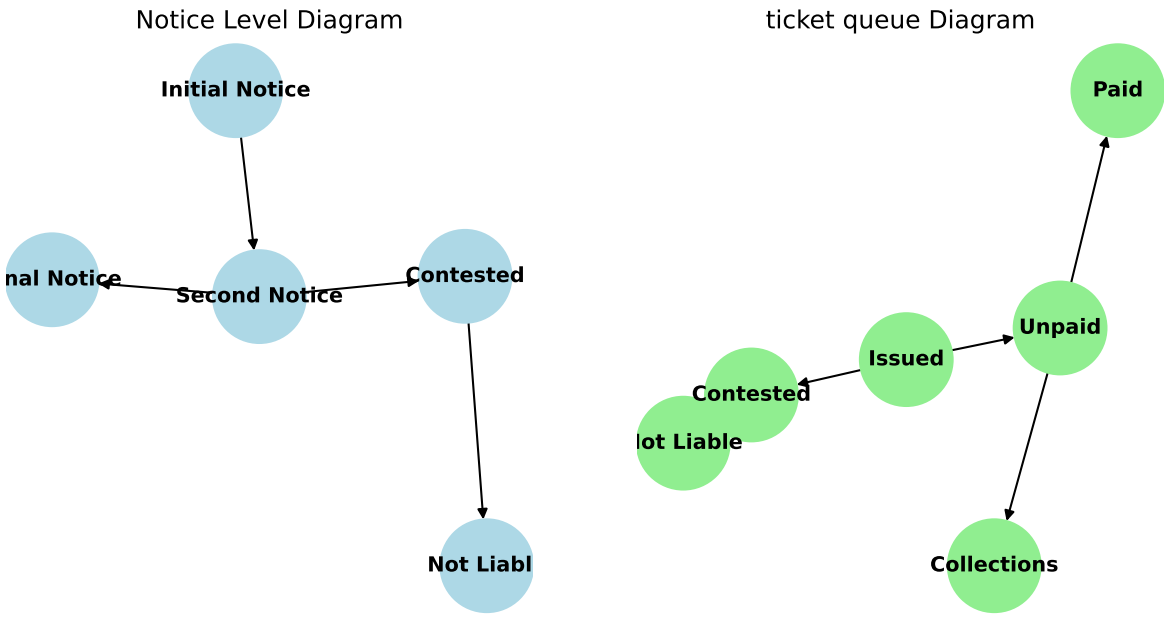
```
# Filter for violations where the fine increases by less than double
df_non_double = df_100[df_100['fine_level2_amount'] < 2 * df_100['fine_level1_amount']]

# Calculate the increase (percentage or absolute) between fine levels
df_non_double['increase'] = (df_non_double['fine_level2_amount'] - df_non_double['fine_level

# Keep only violation_description and the calculated increase
df_non_double_selected = df_non_double[['violation_description', 'increase']]

# Print the result
print(df_non_double_selected)
```

```
                          violation_description  increase
13                        DISABLED PARKING ZONE  0.250000
26                        DISABLED PARKING ZONE  0.250000
40                          PARK OR BLOCK ALLEY  0.666667
46                          PARK OR BLOCK ALLEY  0.666667
67                          PARK OR BLOCK ALLEY  0.666667
...                                         ...       ...
157991  NO CITY STICKER VEHICLE OVER 16,000 LBS.  0.550000
159471  NO CITY STICKER VEHICLE OVER 16,000 LBS.  0.550000
159689  NO CITY STICKER VEHICLE OVER 16,000 LBS.  0.550000
159987  NO CITY STICKER VEHICLE OVER 16,000 LBS.  0.550000
160124  NO CITY STICKER VEHICLE OVER 16,000 LBS.  0.550000

[3976 rows x 2 columns]


/var/folders/b0/6kybjmv57sv8drq7v4kr7w240000gn/T/ipykernel_2628/2526005368.py:5: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  df_non_double['increase'] = (df_non_double['fine_level2_amount'] - df_non_double['fine_leve
```

```python
import networkx as nx

# Create a graph for Notice Level Diagram
notice_graph = nx.DiGraph()

# Add nodes and edges for notice level progression
notice_graph.add_edges_from([
    ('Initial Notice', 'Second Notice'),
    ('Second Notice', 'Final Notice'),
    ('Contested', 'Not Liable'),
])

# Add the conditional path if the ticket is contested and found not liable
notice_graph.add_edge('Second Notice', 'Contested')

# Create a graph for Ticket Queue Diagram
ticket_graph = nx.DiGraph()

# Add nodes and edges for ticket queue progression
ticket_graph.add_edges_from([
```

```
    ('Issued', 'Unpaid'),
    ('Unpaid', 'Collections'),
    ('Contested', 'Not Liable'),
    ('Issued', 'Contested'),
    ('Unpaid', 'Paid'),
])

# Plot Notice Level Diagram
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
nx.draw(notice_graph, with_labels=True, node_color='lightblue', node_size=2000, font_size=10
plt.title("Notice Level Diagram")

# Plot Ticket Queue Diagram
plt.subplot(1, 2, 2)
nx.draw(ticket_graph, with_labels=True, node_color='lightgreen', node_size=2000, font_size=1
plt.title("ticket queue Diagram")
```

```
Text(0.5, 1.0, 'ticket queue Diagram')
```

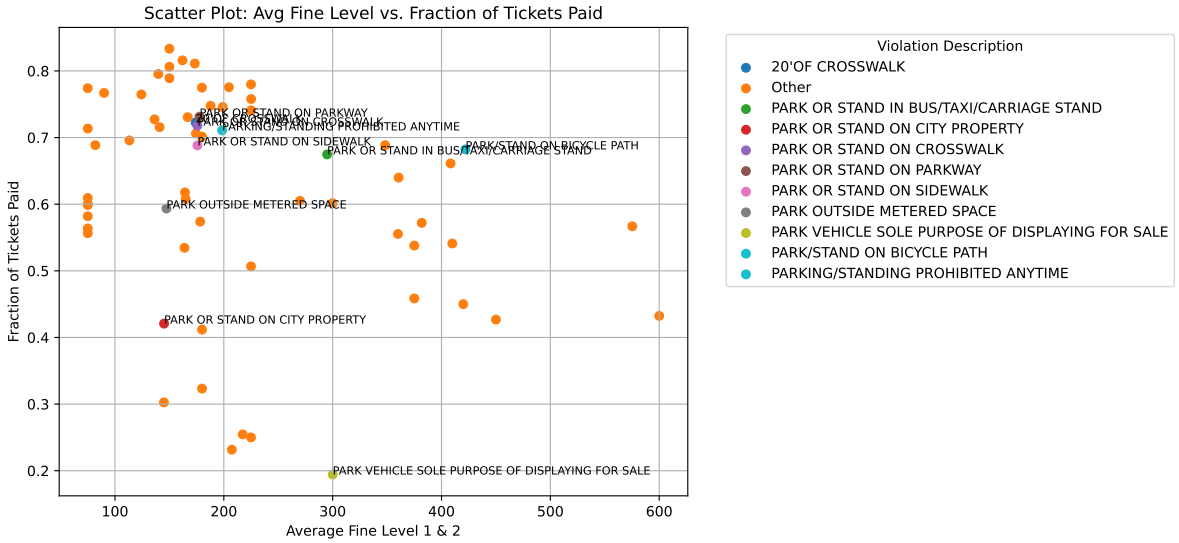


Q3: Resorted help from StackOverflow and Chat GPT

a

```
# Get the top 10 most common violation descriptions
top_violations = grouped2['violation_description'].value_counts().nlargest(10).index

# Create a new column to classify violations
grouped2['violation_category'] = grouped2['violation_description'].apply(
    lambda x: x if x in top_violations else 'Other'
)

# Filter for distinct categories for coloring
unique_categories = grouped2['violation_category'].unique()

# Define a color map for each category
colors = plt.cm.get_cmap('tab10', len(unique_categories))

# Create the scatter plot with labeled dots
plt.figure(figsize=(8, 6))
for i, category in enumerate(unique_categories):
    subset = grouped2[grouped2['violation_category'] == category]
    plt.scatter(subset['avg_fine_level1_2'], subset['fraction_tickets_paid'], label=category

# Add labels for the top 10 violations
for i in range(len(grouped2)):
    row = grouped2.iloc[i]
    if row['violation_category'] != 'Other':
        plt.text(row['avg_fine_level1_2'], row['fraction_tickets_paid'], row['violation_desc

# Add title, labels, and legend
plt.title('Scatter Plot: Avg Fine Level vs. Fraction of Tickets Paid')
plt.xlabel('Average Fine Level 1 & 2')
plt.ylabel('Fraction of Tickets Paid')
plt.grid(True)
plt.legend(title='Violation Description', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.show()
```

```
/var/folders/b0/6kybjmv57sv8drq7v4kr7w240000gn/T/ipykernel_2628/3998723457.py:13: Matplotlibl
  colors = plt.cm.get_cmap('tab10', len(unique_categories))
```



b

```
# Define broader categories for similar violations
def categorize_violation(description):
    if 'parking' in description.lower():
        return 'Parking Violation'
    elif 'expired' in description.lower() or 'meter' in description.lower():
        return 'Expired Meter'
    elif 'loading' in description.lower():
        return 'Loading Zone Violation'
    else:
        return 'Other'

# Apply the function to categorize violations
grouped2['violation_category'] = grouped2['violation_description'].apply(categorize_violatio

# Get unique categories for coloring
unique_categories = grouped2['violation_category'].unique()

# Create a scatter plot with different colors for each category
plt.figure(figsize=(8, 6))
for i, category in enumerate(unique_categories):
    subset = grouped2[grouped2['violation_category'] == category]
    plt.scatter(subset['avg_fine_level1_2'], subset['fraction_tickets_paid'], label=category
```

```
# Add title, labels, and legend
plt.title('Scatter Plot: Avg Fine Level vs. Fraction of Tickets Paid (Categorized)')
plt.xlabel('Average Fine Level 1 & 2')
plt.ylabel('Fraction of Tickets Paid')
plt.grid(True)
plt.legend(title='Violation Category', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.show()
```